INTHELEX: INcremental THEory Learner from EXamples

Teresa Maria Altomare Basile, Bruno Belsanti, Nicola Di Mauro, Antonio Di Palma, Stefano Ferilli
Dipartimento di Informatica – Università degli Studi di Bari
Via E. Orabona, 4 – 70125 Bari
ITALIA

ABSTRACT

INTHELEX is a learning system for the induction of *hierarchical* theories from examples. It is fully and inherently *incremental*, and can learn simultaneously *various concepts*, possibly related to each other. The representation language, which is common to both examples and learned theories, consists of function free Horn clauses to be interpreted according to the Object Identity assumption. This system has been further developed by providing it with additional numeric and multistrategy capabilities, in order to improve effectiveness and efficiency of the learning task. INTHELEX was applied to the real-world domain of document processing, giving encouraging results.

KEY WORDS: Machine Learning, Inductive Logic Programming, Theory Revision, Multistrategy Learning.

1. INTRODUCTION

INTHELEX (INcremental THEory Learner from EXamples) is a learning system for the induction of *hierarchical* theories from positive and negative examples. It is fully and inherently *incremental*: this means that, in addition to the possibility of taking as input a previously generated version of the theory, learning can also start from an empty theory and from the first available example; moreover, at any moment the theory is guaranteed to be correct with respect to all of the examples encountered thus far. Incremental learning is necessary when either incomplete information is available at the time of initial theory generation, or the nature of the concepts evolves dynamically[1]. INTHELEX can learn simultaneously *various concepts*, possibly related to each other and is based on a *closed loop* architecture - i.e. the learned theory correctness is checked on any new example and, in case of failure, a revision process is activated on it, in order to restore completeness and consistency.

[1] The latter situation is the most difficult to handle since time evolution needs to be considered.

INTHELEX learns theories expressed as sets of Datalog^{OI} clauses, i.e. function free clauses to be interpreted according to the Object Identity (OI for short) assumption [2] ("*within a clause, terms denoted with different symbols must be distinct*"). It adopts a *full memory storage* strategy - i.e., it retains all the available examples, thus the learned theories are guaranteed to be valid on the whole set of known examples - and it incorporates two operators, one for generalizing hypotheses that reject positive examples, and the other for specializing hypotheses that explain negative examples. Both these operators, when applied, change the *answer set* of the theory, i.e. the set of examples it accounts for. Therefore, it is a system for theory revision rather than for theory restructuring [7].

Born to handle symbolic representations only, INTHELEX was extended to treat numeric attributes as well. Such an extension was needed because in most domains numeric information is of vital importance. Thus, the representational language was modified in order to allow the presence of both symbolic and numeric constants and relations/properties. In addition, the learned theory can now contain numeric constraints that represent the range of values a numeric variable can assume. Since different numeric attributes may have the same value, the OI-assumption was applied only to symbolic objects, and not to numeric ones.

This purely inductive system has been further developed by providing it with additional multistrategy capabilities [5], according to the Inferential Theory of Learning framework [6], in order to improve effectiveness and efficiency of the learning task. In particular, abduction is used to complete the observations, whenever possible, in such a way that the examples they represent are explained (if positive) or rejected (if negative). This prevents the refinement operators from being applied, as long as possible, leaving the theory unchanged. Abstraction can be cast as the process of focusing on what is relevant in an observation. Indeed, ignoring the details about the objects belonging to a class may facilitate the generation of rules for that class. Deduction exploits the provided Background Knowledge (i.e. some partial concept definitions known to be correct, and hence not modifiable) to recognize known objects in an example description.

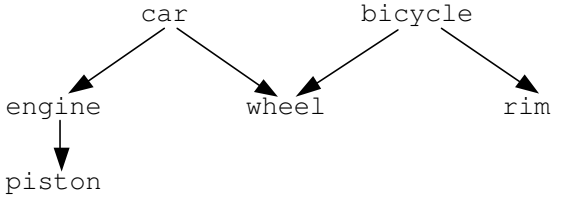

Figure 1. Example of dependence graph.

2. ARCHITECTURE

In order to perform its task, the system exploits a previous theory (optional), a graph describing the dependence relationships among the concepts to be learned (see Figure 1 for a graphical example concerning the vehicle domain), and a historical memory of all the past (positive and negative) examples that led to the current theory. It is important to note that a positive example for a concept is not considered as a negative example for all the other concepts (unless it is explicitly stated). The logical architecture of INTHELEX is shown in Figure 2.

A set of examples of the concepts to be learned is provided by the Expert, possibly selected from the Environment. Whenever a new example is taken into account, it is preliminary processed by the Abstraction module in order to describe it in a higher-level language, by eliminating superfluous details according to the given Abstraction theory. Specifically, the implemented abstraction operators allow to [3]:

- eliminate superfluous details;
- group specific component patterns into compound objects;
- reduce the number of object attributes;
- ignore the number of instances of a given object;
- obtain a coarser grain-size for attribute values.

Then, the example is stored in the historical memory and, if necessary, undergoes a saturation phase. If any of its sub-concepts in the dependency graph can be recognized in its description according to the definitions learned by the system up to that moment and the Background Knowledge, literals concerning those concepts are added (properly instantiated) to the example description.

The whole set of examples can be subdivided into three subsets, namely *training*, *tuning* and *test* examples, according to the way in which examples are exploited during the learning process. Specifically, training examples (if any), previously classified by the Expert, are exploited by the Rule Generator to build a theory that is able to explain them. The initial theory can also be provided by the Expert.

Subsequently, such a theory, plus the Background Knowledge, are checked by the Rule Interpreter against new available examples. The Rule Interpreter takes the set of inductive hypotheses and a tuning/test example as input, and produces a decision. It may exploit a special abductive proof procedure to manage situations in which not only the set of all observations is partially known, but each observation could be incomplete too. In particular, an algorithm by [1], modified as in [4], was adopted in order to hypothesize unknown facts to be added to an observation, provided they are consistent with given integrity constraints. The decision is compared to the correct one and, in case of incorrectness, the cause of the wrong decision can be located.

Test examples are exploited only to check the predictive capabilities of the theory on new observations. Conversely, tuning examples are exploited incrementally by the Rule Refiner to modify incorrect hypotheses according to a data-driven strategy. The Rule Refiner consists of two distinct modules, a Rule Specializer and a Rule Generalizer. In particular, when a positive example is not covered, the Rule Generalizer produces a revised theory obtained in one of the following ways (listed by decreasing priority) such that completeness is restored:

- extending the range of numeric values in a clause and/or replacing a clause in the theory with one of its Least General Generalizations under Object Identity against the problematic example;
- adding a new clause to the theory, obtained by properly turning into variables the constants in the problematic example;
- adding the problematic example as a positive exception.

When, on the other hand, a negative example is covered, the Rule Specializer outputs a revised theory that restores consistency by performing one of the following actions (by decreasing priority):

- restricting the range of numeric values in a clause;
- adding positive literals that are able to characterize all the past positive examples of the concept (and exclude the problematic one) to the clauses that concur to the example coverage (starting from the lowest level possible);
- adding a negative literal that is able to discriminate the problematic example from all the past positive ones to the top-level clause in the theory by which the problematic example is covered;

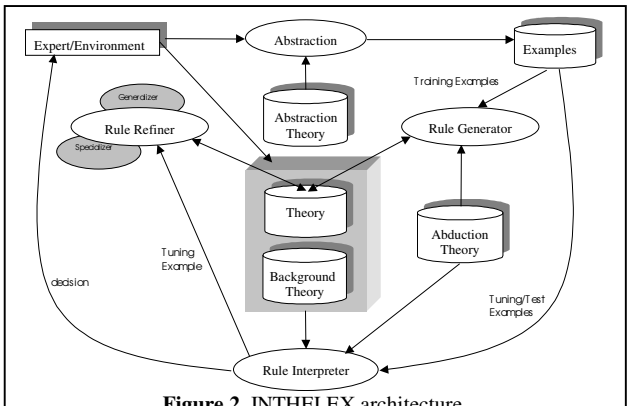

Figure 2. INTHELEX architecture.

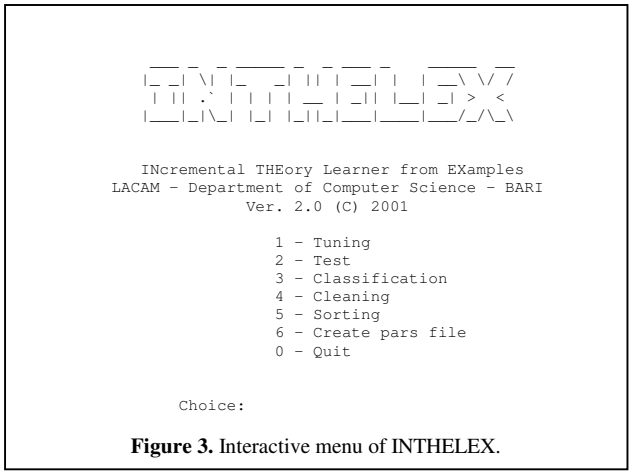

Figure 3. Interactive menu of INTHELEX.

- adding the problematic example as a negative exception.

It is worth noting that INTHELEX never rejects examples, but always refines the theory. Moreover, it does not need to know a priori what is the whole set of concepts to be learned, but it learns a new concept as soon as examples about it are available.

3. RUNNING THE SYSTEM

INTHELEX has been developed in SICSTUS Prolog on a Sun UltraSparc2 architecture running the OPEN WINDOWS environment and on a iX86 architecture running the MS WINDOWS 98 environment. The TCL/TK 8.2 library is exploited to provide the visual interface.

The system accepts examples of the concepts to be learned, described as ground Horn clauses whose head identifies the object and its correct classification, while the body describes the observation in terms of the predicates belonging to the description language (negative literals are not allowed):

```
car(c1) :- engine(c1,m1), wheel(c1,w1),
   wheel(c1,w2), wheel(c1,w3), wheel(c1,w4).
```

According to intuition, a negative example differs from a positive one in that the concept in its head is negated:

```
not(car(c2)) :- engine(c2,m2), wheel(c2,w1),
   wheel(c2,w2), has_pedals(c2).
```

Analogously, each learned hypothesis consists of a Horn clause without symbolic constants, whose head identifies the concept it defines, while the body reports the conditions that an observation should meet in order to be classified in that way. In this case, negative literals (coming from the specialization operator) can be present:

```
car(X) :- engine(X,Y), wheel(X,Z),
   not(has_pedals(X)).
```

Note that, according to the Object Identity assumption, different variables are automatically considered by the system as referring to different objects.

A number of parameters is provided in order to allow or exclude multistrategy and/or limit the search space of the refinement operators. The user can also set the system to autonomously modify, within a controlled range, some of these parameters in order to best fit the specific situations that may take place during learning.

INTHELEX can be run in one of three modes, according to the way the interaction with the user takes place. In particular, the available modes are:

- *Interactive*: The user is asked to directly enter through a textual menu-driven interface (depicted in Figure 3) the parameters according to which the system has to perform its task.
- *Automatic*: The parameters are read from a proper file, and hence the user is not asked for them (this is useful when running many experiments with the same parameters, or if an external application exploits INTHELEX as its learning component).
- *Visual*: It is possible enter the parameters directly through the interface or to load them from a file. Figure 4 shows the visual interfaces for the Tuning and the Test tasks.

The system reports about the actions taken during its execution, and ends with a statistical summary (see Figure 5); such outputs can be displayed on the screen or redirected to any other user-specified device.

4. EXPERIMENTAL RESULTS

INTHELEX was applied to various different tasks. In particular, it can be interesting to report the results obtained on the real-world problem of paper document classification and interpretation. The dataset consisted of 112 scientific papers, 30 of class Springer-Verlag Lecture Notes (SVLN), 34 of IEEE Transactions (IEEET) and 28 of Proceedings of the International Conference on Machine Learning (ICML); other 20 belonged to class Reject. After a preliminarily digitalization and processing in order to describe their layout structure in our representation language, the dataset was randomly split 33 times into learning and test sets (70% and 30% of the whole, respectively). Abstraction was used to shift from punctual values for numeric attributes to symbolic constants representing intervals.

A first experiment aimed at learning rules for classifying documents, and resulted in a 90.73% average predictive accuracy. Another one tried to learn definitions for the logical components of each class (e.g. *title*, *abstract*, etc.). In this case the average results were 92.82% for SVLN, 96.7% for ICML and 96.72% for IEEET.

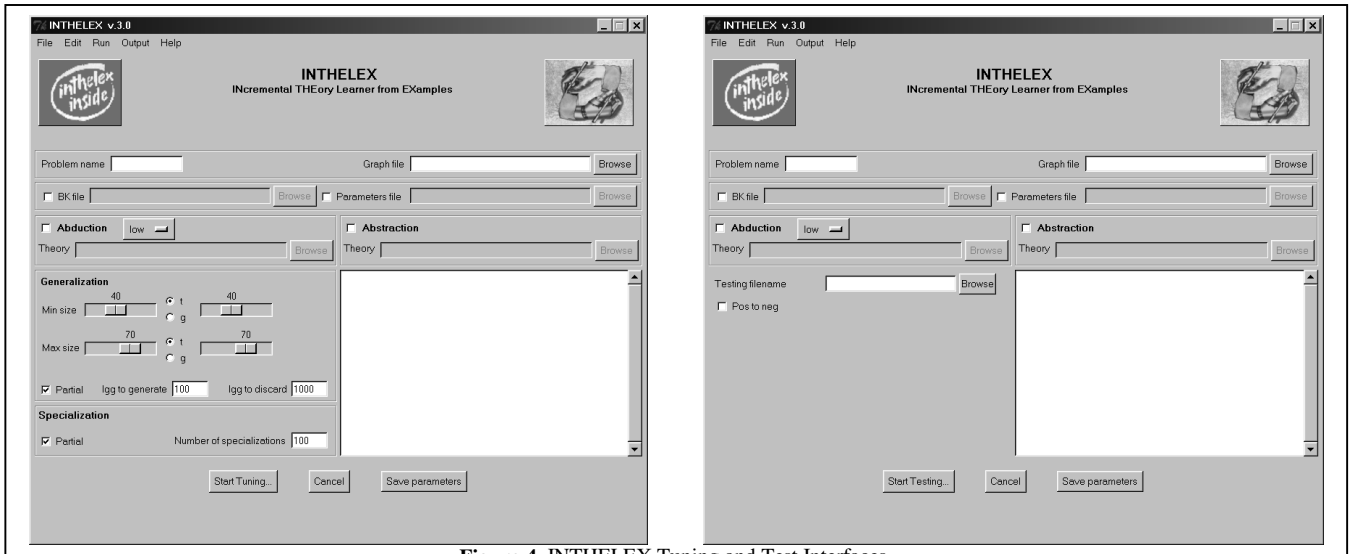

Figure 4. INTHELEX Tuning and Test Interfaces.

5. CONCLUSION

The incremental learning system INTHELEX has been presented, that embeds operators for theory revision and is able to handle both symbolic and numeric information. Its multistrategy capabilities rely on abstraction, abduction and deduction to improve efficiency and effectiveness of the learning process. The use of INTHELEX is made easier thanks to an intuitive representation language and a visual interface through which parameters can be set and modified.

The goodness of the implemented approach has been tested on, and confirmed by, system performance on the real-world domain of paper document processing.

REFERENCES

Proceedings Papers:

[1] A.C. Kakas & P. Bancarella, On the relation of truth maintenance and abduction, *Proc. 1st Pacific Rim International Conference on Artificial Intelligence*, Nagoya, Japan, 1990.

[2] G. Semeraro, F. Esposito & D. Malerba, Ideal refinement of datalog programs, *Logic Program Synthesis and Transformation, Lecture Notes in Computer Science 1048*, 1996, 120-136.

[3] J.-D. Zucker, A semantic abstraction for concept representation and learning, *Proc. 4th International Workshop on Multistrategy Learning*, Desenzano del Garda, Italy,1998.

Journal Papers:

[4] F. Esposito, G. Semeraro, N. Fanizzi & S. Ferilli, Multistrategy theory revision: induction and abduction in INTHELEX, *Machine Learning Journal, 38*(1/2), 2000, 133-156.

Books:

[5] S. Ferilli, *A framework for incremetal synthesis of logic theories: an application to document processing* (PhD Thesis, Dipartimento di Informatica - Università degli Studi di Bari, 2000).

[6] R.S. Michalski, *Inferential theory of learning, developing foundations for multistrategy learning* (In R. S. Michalski & G. Tecuci, editors, Machine Learning. A Multistrategy Approach, volume IV, pages 3-61. Morgan Kaufmann, San Mateo, CA, USA, 1994).

[7] S. Wrobel. *First order theory refinement* (de Raedt, editor. Advances in Inductive Logic Programming. IOS Press, Amsterdam, NL, pages 14-33, 1996).

```
 RESEARCH          :p
 MIN SIZE RATE     :0.5
 MAX SIZE RATE     :0.7
 MAXGEN            :50
 MAX_DISC_CLAUSES  :500

 RESEARCH_SPEC     :p
 MAXSPEC           :99

Time elapsed for tuning: Runtime : 36.200 sec.

Positive example : 20 Negative example : 59

Generalization:
   Added new clause          : 4
   Generated lgg of a clause : 4
   Added positive exception  : 0

Specialization:
   Added positive literal    : 1
   Added negative literal    : 0
   Added negative exception  : 0
```

Figure 5. Output statistics about a Tuning task.